Agile attitude

Review of agile methods for use in design education

Ovesen, Nis; Eriksen, Kaare; Tollestrup, Christian

Published in:
Design Education for Creativity and Business Innovation

Publication date:
2011

Document Version
Publisher's PDF, also known as Version of record

INTERNATIONAL CONFERENCE ON ENGINEERING AND PRODUCT DESIGN EDUCATION
8 & 9 SEPTEMBER 2011, CITY UNIVERSITY, LONDON, UK

AGILE ATTITUDE: REVIEW OF AGILE METHODS FOR USE IN DESIGN EDUCATION

Nis OVESEN, Kaare ERIKSEN and Christian TOLLESTRUP
Department of Architecture, Design & Media Technology, Aalborg University, Denmark

ABSTRACT
Motivated by the growing number of hybrid products with extensive software and service components, this paper reviews four representative agile methods from the domain of software for their applicability to both the industrial design education and practice. The review is based on a comparative analysis of the methods through four scopes, including their application spectrum, level of guidance, process coverage and project management capabilities. Based on the analysis, it is argued that agile methods are able to offer design students the tools for bridging the gap between the software development teams and their own domain of physical product development.

Keywords: Agile development, design education, design methodology, physical product development vs. software

1 INTRODUCTION

With the vibrant array of technological breakthroughs surfacing in the recent years, numerous new product categories are developed. The products – often equipped with embedded software and operating systems, near-field-communication, geo-positioning and may other technologies – are to a great extend hybrids between physical products and non-physical software or services. As this trend seems to be growing, it seems appropriate to consider whether or not design students should be equipped with the corresponding vocabulary and catalogue of methods that bridge the gap and enables students to take part in these new product development activities after graduation.
During the last ten years *Agile Development* has gained a lot of attention as a solution to the often complex development projects within the software industry. This paper attempts to clarify the benefits of Agile Development in relation to the design profession and as a potential addition to the curriculum at the design educations.
During the last couple of years, a few projects have been aiming at bringing agile methods into the domain of physical product design and development. Two main contributions are, firstly "Agile Project Management" [1] by Jim Highsmith – one of the founding fathers of the Agile Development concept, and secondly "Flexible Product Development" [2] by Preston G. Smith. Both contributions aim at implementing agile development concepts in physical product development, but primarily in large-scale companies and with a focus on management aspects.
The present paper presents an interpretation of agile methods with the scope of the industrial designer. As part of this interpretation is a comparative analysis of four selected agile methods and their applicability to industrial design education as well as considerations about the basic differences and similarities between physical product development and software development.
The rest of this paper is composed as follows. The second sections present a brief overview of four agile methods and their common background. The third section presents the four "scopes," preparing for the comparative method analysis, which comprises the fourth section. Lastly, the fifth section sets a discussion about the applicability of agile methods to the design education.

2 AGILE METHODS

Agile is an attribute often associated with animals like the big cats. Alert and responsive – quick and well coordinated in movement [3]. Nevertheless, these are also the features that can be associated with agile methods in a process management perspective. In the following, an overview of four recognised agile methods and their specific characteristics is presented, but before this, a short and general overview of Agile Development is outlined.

2.1 Agile values

As a term, Agile Development was coined in 2001 [4] in *The Manifesto for Agile Software Development.* The manifesto precisely states that agilists value "Individuals and interactions over processes and tools;" "Working software over comprehensive documentation;" "Customer collaboration over contract negotiation" and "Responding to change over following a plan" [5].

2.2 Four agile methods

As seen in the manifesto, Agile Development is a set of values rather than actual methods. However, a span of various methods that each in some way proposes new ways of undertaking development activities, seem to apply to those values, and are therefore called agile methods. In general, agile methods are a subset of iterative methods [6] and often categorised as "light weight", meaning low on documentation requirements and able to accommodate changing project surroundings. The list of methods claiming to be agile is long, thus four representative methods are outlined in the following.

2.2.1 Scrum

The term Scrum originates from Rugby for getting a ball into play. The name was chosen because of the similarities between this. Scrum emphasises an empirical process rather than a defined process [6]. The lifecycle of the scrum process roughly divided in three defined phases: pre-game, development, and post-game.

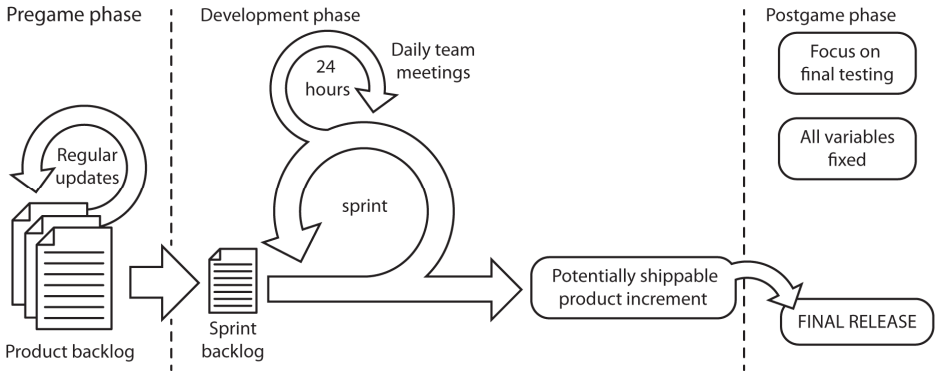

Figure 1. Scrum process with three phases

The pre-game phase includes the development of an initial project specification, which is often mentioned as the product backlog. The backlog contains all requirements currently known from customers, sales and marketing division, customer support and developers [7]. The development phase is the agile part of scrum. Rather than ultimately deciding variables such as requirements, resources, technologies, and tools only at the beginning of a project, the development phase is organised in short iterative cycles called *sprints*, where these variables are continuously revised and thoroughly controlled. A sprint focuses on development of only a few collectively chosen features in the product backlog list. Scrum emphasises self-organising teams and daily scrum-meetings between all team members. Each sprint ends with a sprint review and a revision of the backlog [7].
Scrum seams to leave the actual methods in the practical development activities up to the team and is thereby mostly a tool for managing the development process rather than an actual development method.

2.2.2 Extreme Programming (XP)

Extreme Programming (XP) was coined as a distinct development method by combining a number of best practices in software development. XP is based on the conviction that it is hard to have too much of a good thing [6] and is founded on the four values communication, simplicity, feedback, and courage. Beside these values, XP recommends some core practices [2] shown in Table 1:

Table 1. List of Extreme Programming core practices [2].

1. Planning game	5. Testing
2. Small, frequent releases	6. Pair programming
3. Project metaphor	7. Team code ownership
4. Simple design	

The process of XP consists of five phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance and Death. Extreme Programming emphasises iterative and incremental development just as Scrum does it, but in contrast to Scrum, XP provides explicit and hands-on methods for developers. It is probably also the reason why XP is so broadly adopted as a practical and relevant technique [6]. Some of those hands-on practices taken to the extreme are:

- Testing is good, so write test before writing the code
- Short iterations and early feedback are good, so make iterations shorter – one to three weeks
- Customer collaboration is good, so have customers in the team fulltime.

Extreme Programming is clearly meant as a tool for software development teams, but because of its explicit and straightforward practices, it might be fruitful to see it in relation to physical product development as well.

2.2.3 Feature-driven Development (FDD)

In opposition to Extreme Programming Feature-driven Development does not provide concrete guidance in respect to specific development methods. It is mostly a management-supporting tool that suggests a specific framing of the process as well as iterative development in a certain way. Feature-driven Development consists of five chronological processes: *Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature, and Build by Feature* [7]. The last two processes run in an iterative cycle, and therefore are changes to product requirements and business needs possible even late in the overall process. FDD proposes fast iterative cycles between one and three weeks with focus on only one or few features at a time. As the name also suggests, Feature-driven Development is based on the precondition that the work-product can be split into more or less independent parts, which is most often possible in software projects.

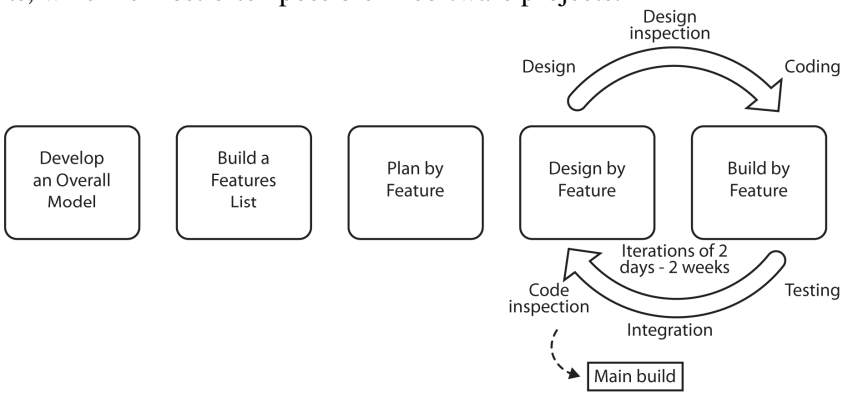

Figure 2. FDD process lifecycle [7].

FDD is emphasising quality all the way through the process lifecycle and aims at frequent deliveries [8].

2.2.4 Pragmatic Programming (PP)

Pragmatic programming is not a specific method as such, as it does not refer to the design process as a whole, nor has it been described as a specific method for software development. Nevertheless, the name Pragmatic Programming is typically used to summarise a total of 70 short tips and best practices described in the book "The Pragmatic Programmer" by Andrew Hunt and David Thomas [7]. The authors call it "an attitude, a style, a philosophy of approaching problems and their solutions" [9], and emphasise the philosophy in six points:

1. **Take responsibility.** You make a commitment to ensure that something is done right, but you don't necessarily have direct control over every aspect of it.
2. **Software Entropy.** Don't leave "broken windows" (bad designs, wrong decisions, or poor code) unrepaired.
3. **Be a catalyst for change.** Don't be like the frog in the boiling water. Keep an eye on the big picture. Constantly review what's happening around you.
4. **Good-enough software.** Great software today is often preferable to perfect software tomorrow.
5. **Update your knowledge portfolio.** Keep yourself updated with the needed knowledge in your field, and critically analyse what you read and hear.

6. **Communicate!** Improve your communication skills - A good idea is an orphan without effective communication.

(Reworked after Hunt & Thomas) [9]

As mentioned above, the authors call Pragmatic Programming an attitude, and this seems to be the right way to describe it. Pragmatic Programming is an explication of common sense and good practice in development activities and brings the discussion about development to a day-to-day level without prescribing a great master plan of the whole development process.

3 THE SCOPES IN THE ANALYSIS

Before continuing to the comparative analysis, the four scopes in the analysis are listed below and presented in the following.

Table 2. The four scopes

The four scopes
1. Software specific vs. general application
2. Abstract principles or concrete guidance
3. Coverage of product development process
4. Applicable process management tools

'Software specific versus general application' aims at evaluating the various agile methods with respect to their applicability to other areas than software development and related programming activities, specifically Industrial Design.

'Abstract principles or concrete guidance' compare the various methods with respect to their ability to provide concrete day-to-day guidance to design students on specific challenges or problems inevitably rising in development projects.

The scope *'Coverage of product development process'* evaluates the respective agile methods' abilities to cover the various parts of the product development lifecycle. The coverage is related to a generic product design and development process model in order to visualise their respective strengths.

The scope *'Applicable project management tools'* is for scanning the agile methods for project managing tools. The assumption behind this scope is that project management tools often operate on a higher level than the domain-specific techniques and practices such as software programming, physical product construction or design, and will therefore be relevant to design students.

4 COMPARATIVE ANALYSIS OF AGILE METHODS

In this section, the agile methods outlined in section 2 are compared using the analytical scopes defined above. Whereas the four scopes are evaluated separately, the graphical figures in this section build broadly on all scopes.

4.1 Software specific vs. general application

Scrum concentrates on team member interaction and communication in order to facilitate project flexibility in constantly changing environments. Emphasis is not on specific software programming techniques, and it can therefore be easily implemented into student projects.

Extreme Programming is possibly the most software specific method reviewed in this study as it proposes certain programming and debugging routines and other software related practices. Despite this are many of the practices, such as project metaphors, simple design, team ownership, and extensive testing general recommendations, which can easily be transferred to physical product development or a learning environment. As can the four values *communication*, *simplicity*, *feedback*, and *courage*.

Feature-driven Development emphasises quality and frequent deliveries through short iterations, but does not recommend any specific techniques that binds it to software except from the precondition of splitting up the product specification into separate "features" that can be individually developed. If this is possible in a given design project, FFD is relevant to more than just software.

Being promoted as an attitude, Pragmatic Programming comprises a set of values that are directly transferrable to other industries than software, but the many best practices and tips focuses on software specific programming elements. In short PP can be said to offer general guidance to establish the right agile mindset for both students and professionals.

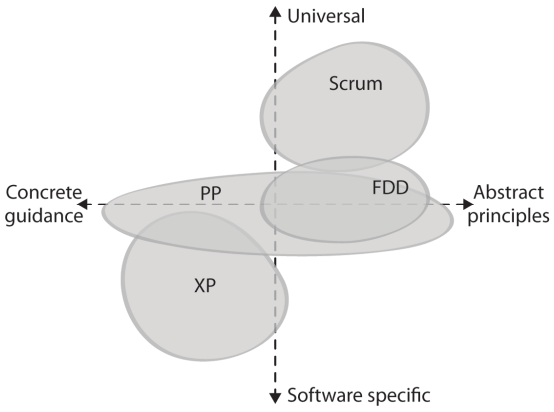

Figure 3. Software specific vs. Universal application – concrete guidance vs. abstract principles.

4.2 Abstract principles or concrete guidance

When it comes to providing day-to-day concrete guidance on for instance choice of techniques, Scrum might be one on the least informative. This may be problematic in a learning environment. But as mentioned earlier, Scrum offers certain principles such as a management framework with daily scrum meetings – not information on how to specifically perform a certain task. Extreme Programming is somewhat opposite as it is high on concrete guidance and procedures. Feature-driven Development offers no concrete methods to be used for the specific process steps, but just as Scrum does FDD have certain management-supporting principles. Pragmatic Programming provides many day-to-day advices through the 70 tips it consists of, but it does, however, also take the guidance to a more principle level in its discussion about the right agile attitude.

In general, the abstract guidance seems easily transferrable to the domain of physical product design and development in the sense that this type of guidance is on such a general level, that the domain barriers are somewhat insignificant. Concrete guidance, however, can be difficult to transfer to physical product development without modifying it to fit into the restraints of this domain.

4.3 Coverage of product development process

This scope will compare the methodological possible coverage of the product development process. As reference is the Generic Product Development Process by Ulrich and Eppinger [10].

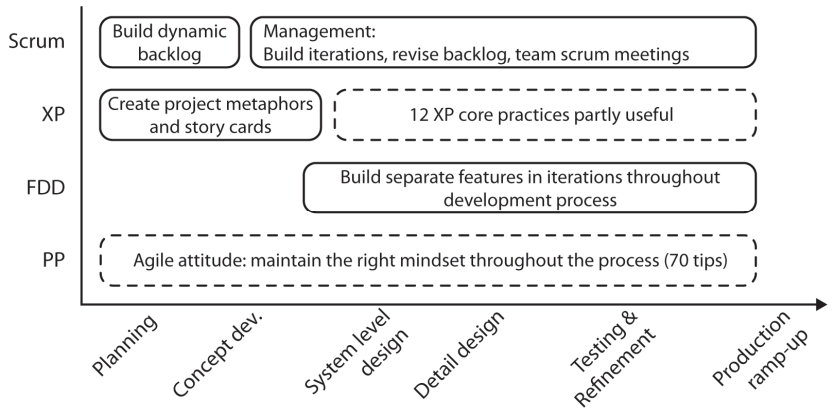

Figure 4. Process lifecycle with the relevant aspects of the four methods.

As shown in Figure 4 above, all four methods are considered to have elements that relates to the design and development process in industrial design. This means that design students would be able to meaningfully implement elements of the methods in their project work.

4.4 Applicable process management tools

From the descriptions of the agile methods, we have seen that process management is as important a part as the concrete day-to-day practices and techniques. These process tools can to a large extent be

used as general practices. In particular Scrum and Feature-driven Development include this kind of prescriptions, such as

- Development in iterative cycles, timeboxed from one to six or eight weeks per cycle.
- Continuous and formalised revision of the initial project specification.
- Self-organising teams collectively decides which product increments to work on during a cycle.
- Daily scrum meetings as a team communication tools.

5 DISCUSSION

This paper has reviewed four agile methods for an evaluation of their applicability to stretch beyond the domain of software. As outcome of this method review, we have learned that agile methods seem to span from concrete day-to-day guidance to abstract principles about team mentality etc. It is argued that all four presented methods have elements that are useful to physical product design and development activities, thus making them applicable as methods for the new wave of hybrid products. As the domains of software and product development and design are merging in modern products, it is arguably important to promote common terminologies and methods to design students in order to make this integration between software development and physical product design happen as seamlessly as possible.

5.1 The right learning mindset

Agile developers refer to The Agile Value Set, emphasising individuals and interactions, working prototypes (products), close customer collaborations, and responding to change. Altogether these values reflect an attitude towards product development and the business environment surrounding them. “Agile is an attitude”, as the authors of Pragmatic Programming state; it is about formalising “common sense” and “good practices”. Suscheck & Ford goes as far as to use *jazz improvisation* as a metaphor, claiming that it better elucidates the supportive organisational culture required to effectively use agile development processes [11].

This paper is initiated by the need for domain-bridging methods that enables the design of hybrid products. However, through the investigation of agile methods, it has become clear, that the agile mindset seems beneficial in almost any situation. With development trends going towards faster development and need for high flexibility, and market trends going towards high uncertainty, it seems evident that design students will need the competences of cross-disciplinary communication, flexibility and readiness for all sorts of change. An agile mindset seems to be a way to achieve those competences.

REFERENCES

[1] Highsmith, J., *Agile Project Management – Creating Innovative Products, 2nd ed.*, 2010 (Addison-Wesley, Boston).

[2] Smith, Preston G. Flexible Product Development: Building Agility for Changing Markets, 2007 (John Wiley & Sons, California).

[3] *Agile definition from dictionary.com*, http://dictionary.reference.com/browse/agile, 21-12-2010.

[4] *History: The Agile Manifesto*, http://agilemanifesto.org/history.html, 21-12-2010.

[5] *Manifesto for Agile Software Development*, agilemanifesto.org, 21-12-2010.

[6] Larman, C., *Agile and Iterative Development: A Manager's Guide*, 2004 (Addison-Wesley Professional, Boston).

[7] Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J., *Agile Software Development Methods – Review and analysis*, 2002 (VTT Publications, Finland).

[8] Abrahamsson, P., Warsta, J., Siponen, M.T. and Ronkainen, J., New Directions on Agile Methods: A Comparative Analysis, in *Proc. 25th International Conference on Software Engineering, ICSE’03*, May 2003, pp244-254.

[9] Hunt, A. and Thomas, D., *The Pragmatic Programmer: From Journeyman to Master*, 1999 (Addison-Wesley, Boston).

[10] Ulrich, K.T. and Eppinger, S.D, *Product Design and Development, 3rd ed.*, 2003 (McGrag-Hill, Singapore).

[11] Suscheck, C.A. and Ford, R., Jazz Improvisation as a Learning Metaphor for the Scrum Software Development Methodology, in *Software Process Improvement and Practice,* June 2008, 13, pp 439-450.